Augmenting Data Structures

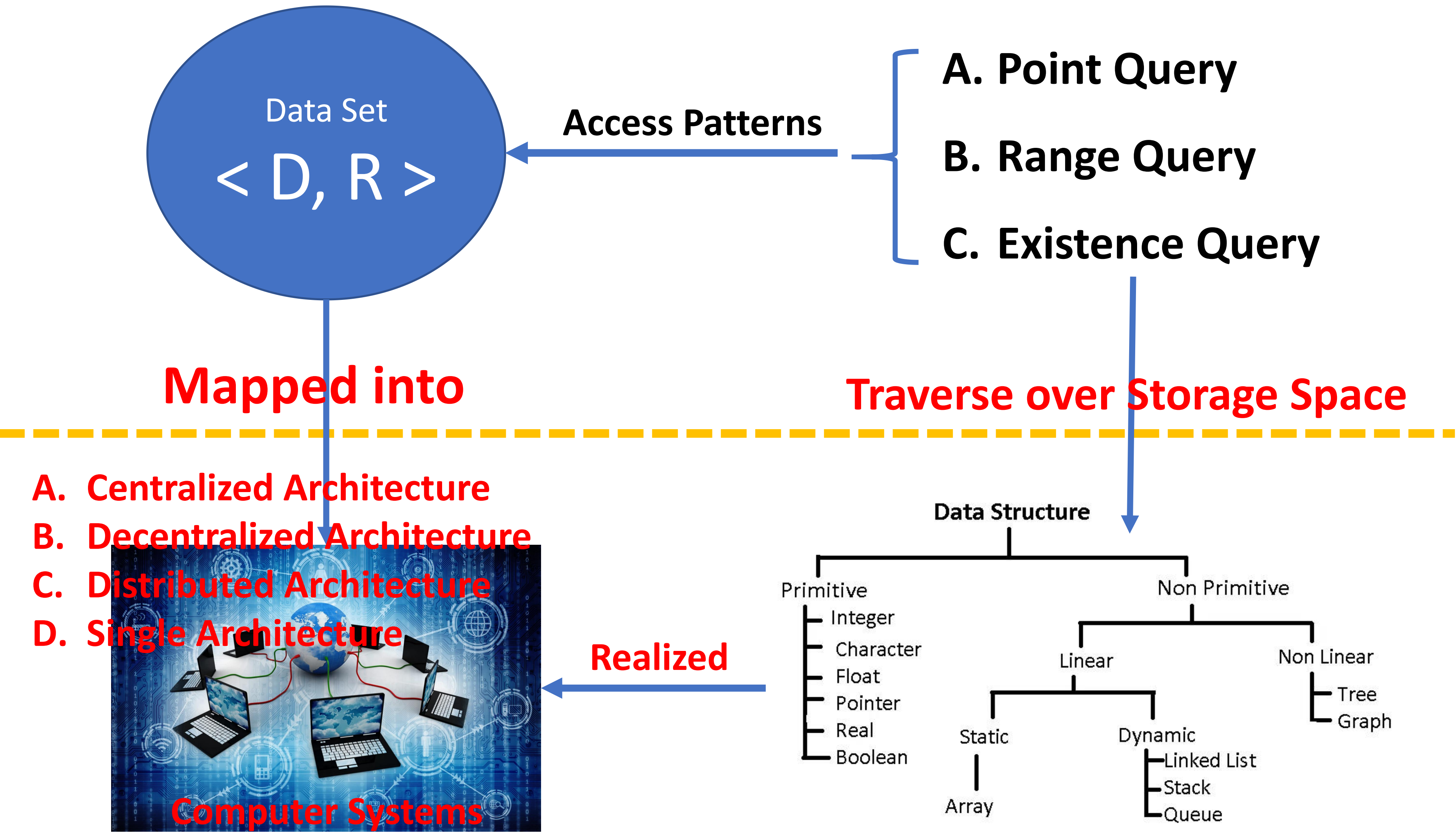
Zhihong Chong

https://cs.seu.edu.cn/chongzhihong

Outline

- **Why Augmenting Data Structures**
  - Access Pattern: Ranked Query
  - Non-Ordered Data: Data of Intervals
- **B+ based Augmenting Data Structures**
  - R-B+
  - Hilbert-Tree
- **Applications and Problems**

Framework of Data Structures



Framework of Data Structures

Data Set

$< D, R >$

Access Patterns

A. Point Query

B. Range Query

C. Existence Query

What are Data Structures?

A Particular way of organizing data set into computer systems so that its access is efficient, usually at the cost of sublinear complexity

Mapped into

Traverse over Storage Space

A. Centralized Architecture

B. Decentralized Architecture

C. Distributed Architecture

D. Single Architecture

Realized

Data Structure

- Primitive
  - Integer
  - Character
  - Float
  - Pointer
  - Real
  - Boolean
- Non Primitive
  - Linear
    - Static
      - Array
    - Dynamic
      - Linked List
      - Stack
      - Queue
  - Non Linear
    - Tree
    - Graph

Retrieval for knowledge-intensive NLP tasks

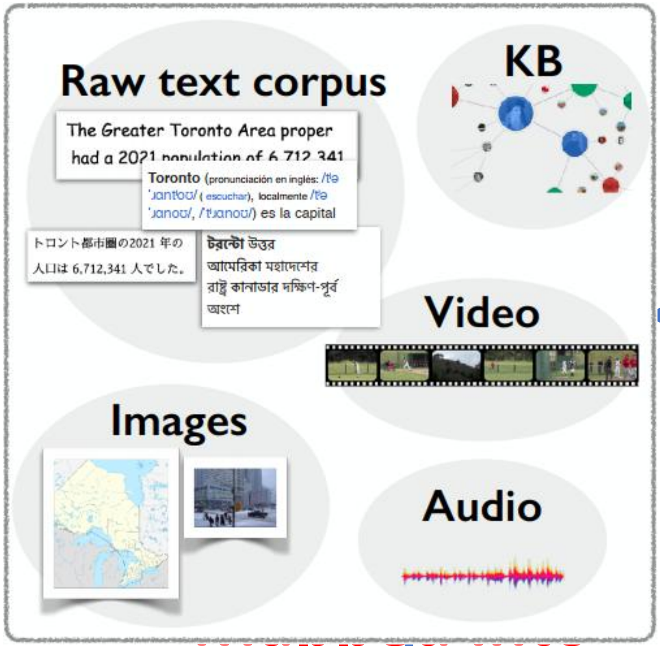
Representative tasks: open-domain QA, fact checking, entity linking, ..

Image: http://ai.stanford.edu/blog/retrieval-based-NLP/

Drives a lot of research on better algorithms for **dense retrieval**, e.g., **DPR** (Karpukhin et al., 2020), **ColBERT** (Khattab and Zaharia, 2020), **ANCE** (Xiong et al., 2021), **Contriever** (Izacard et al., 2022), …

Why retrieval → LMs?

Framework of Data Structures



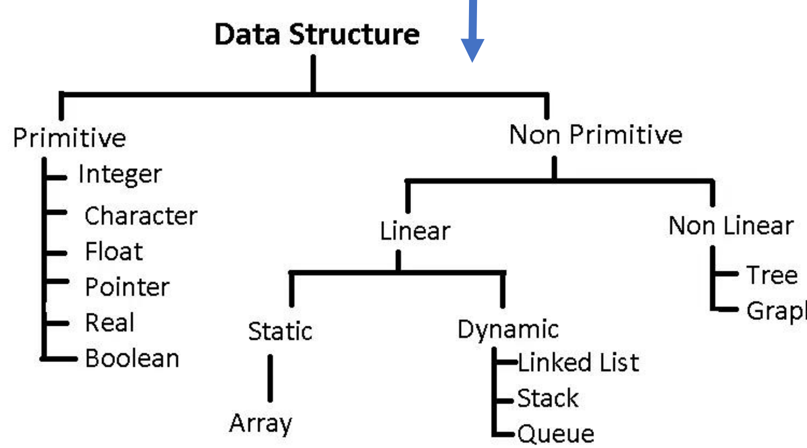
Access Patterns

A. Point Query
B. Range Query
C. Existence Query

Traverse over Storage Space

A. Centralized Architecture
B. Decentralized Architecture
C. Distributed Architecture
D. Single Architecture

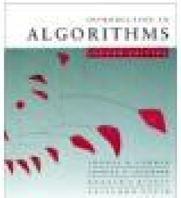
Realized



Outline

- **Why Augmenting Data Structures**
  - Access Pattern: Ranked Query
  - Non-Ordered Data: Data of Intervals
- **B+ based Augmenting Data Structures**
  - R-B+
  - Hilbert-Tree
- **Applications and Problems**

Why Augmenting Data Structures



Dynamic order statistics

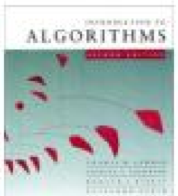
OS-SELECT(i, S): returns the ith smallest element in the dynamic set S.

OS-RANK(x, S): returns the rank of $x \in S$ in the sorted order of S's elements.

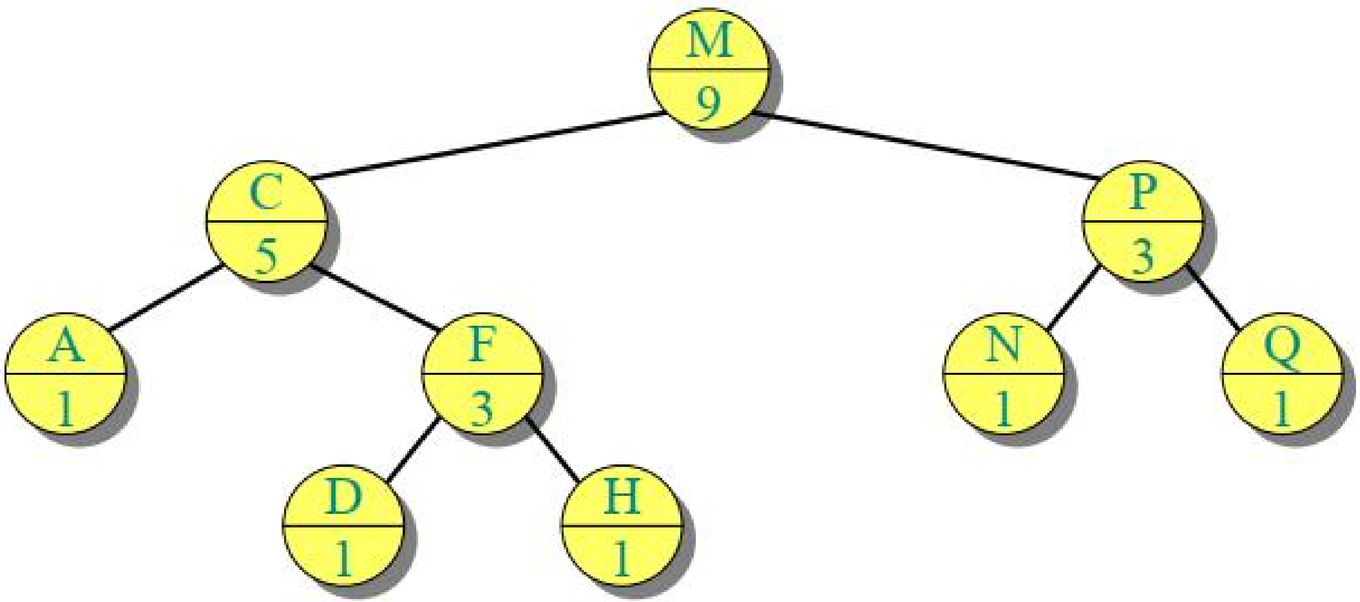
IDEA: Use a red-black tree for the set S, but keep subtree sizes in the nodes.

Notation for nodes: 

Why Augmenting Data Structures

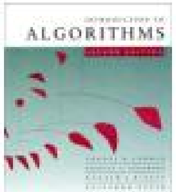


Example of an OS-tree



$$size[x] = size[left[x]] + size[right[x]] + 1$$

Why Augmenting Data Structures



Selection

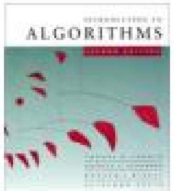
Implementation trick: Use a ***sentinel*** (dummy record) for NIL such that $size[\text{NIL}] = 0$.

OS-SELECT(x, i) ▷ ith smallest element in the subtree rooted at x

$k \leftarrow size[left[x]] + 1$ ▷ $k = \text{rank}(x)$
if $i = k$ **then return** x
if $i < k$
  then return OS-SELECT$(left[x], i)$
  else return OS-SELECT$(right[x], i - k)$

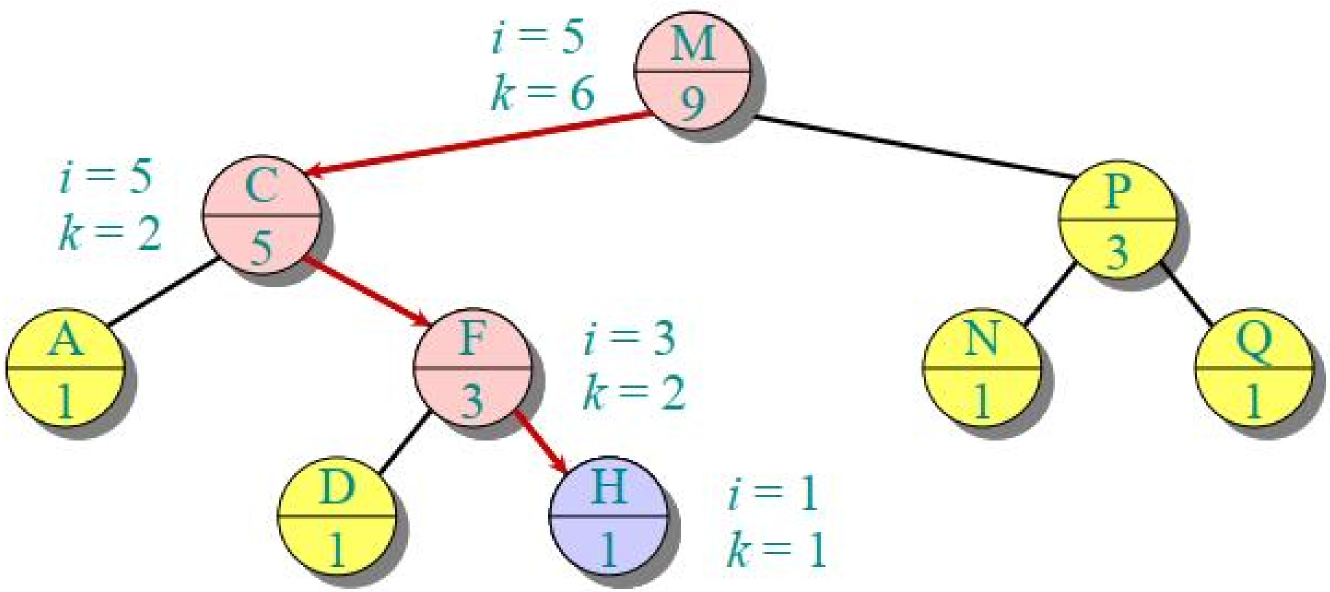
(OS-RANK is in the textbook.)

Why Augmenting Data Structures



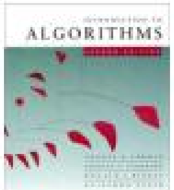
Example

OS-SELECT(*root*, 5)



Running time = $O(h) = O(\lg n)$ for red-black trees.

Why Augmenting Data Structures



Data structure maintenance

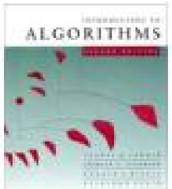
Q. Why not keep the ranks themselves in the nodes instead of subtree sizes?

A. They are hard to maintain when the red-black tree is modified.

Modifying operations: INSERT and DELETE.

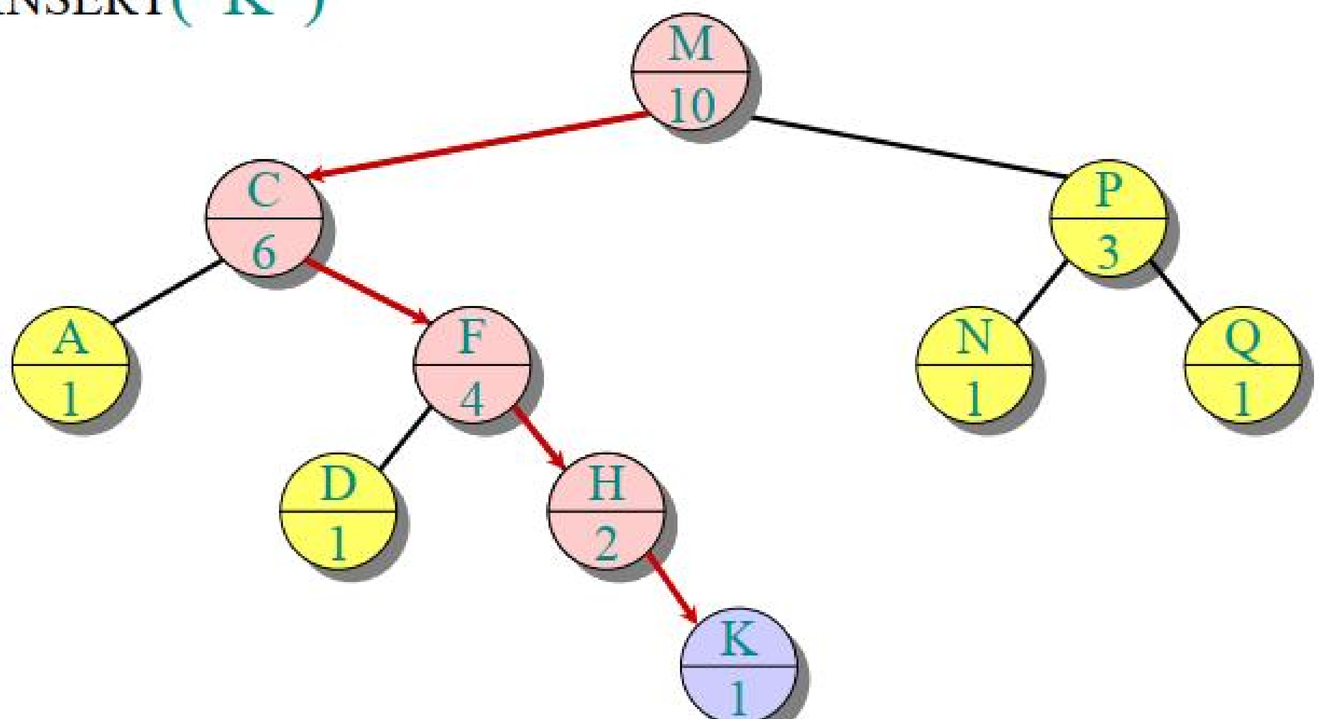
Strategy: Update subtree sizes when inserting or deleting.

Why Augmenting Data Structures

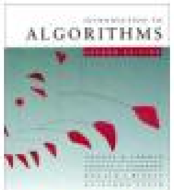


Example of insertion

INSERT(“K”)



Why Augmenting Data Structures

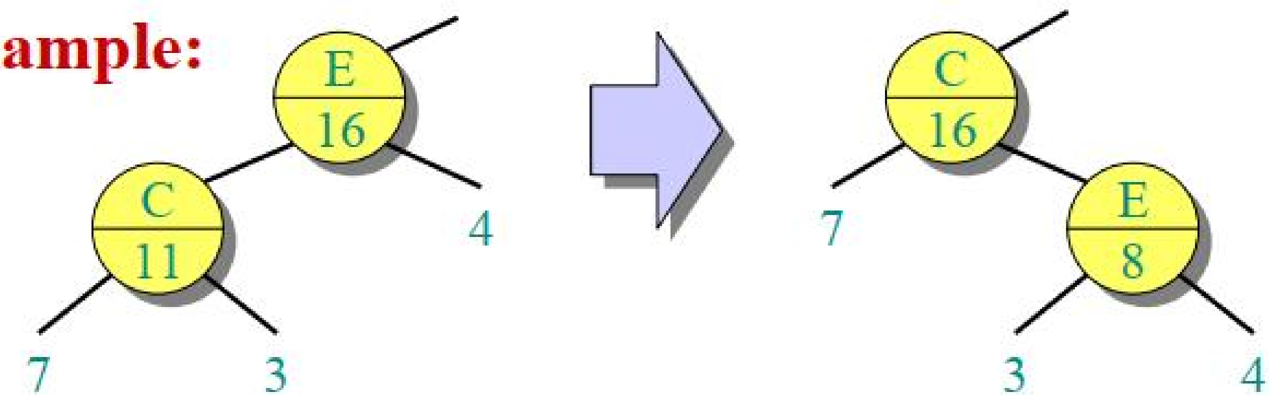


Handling rebalancing

Don't forget that RB-INSERT and RB-DELETE may also need to modify the red-black tree in order to maintain balance.

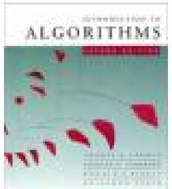
- *Recolorings*: no effect on subtree sizes.
- *Rotations*: fix up subtree sizes in $O(1)$ time.

Example:



∴ RB-INSERT and RB-DELETE still run in $O(\lg n)$ time.

Why Augmenting Data Structures



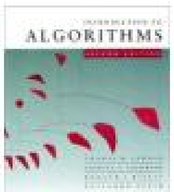
Data-structure augmentation

Methodology: (*e.g., order-statistics trees*)

1. Choose an underlying data structure (*red-black trees*).
2. Determine additional information to be stored in the data structure (*subtree sizes*).
3. Verify that this information can be maintained for modifying operations (*RB-INSERT, RB-DELETE — don't forget rotations*).
4. Develop new dynamic-set operations that use the information (*OS-SELECT and OS-RANK*).

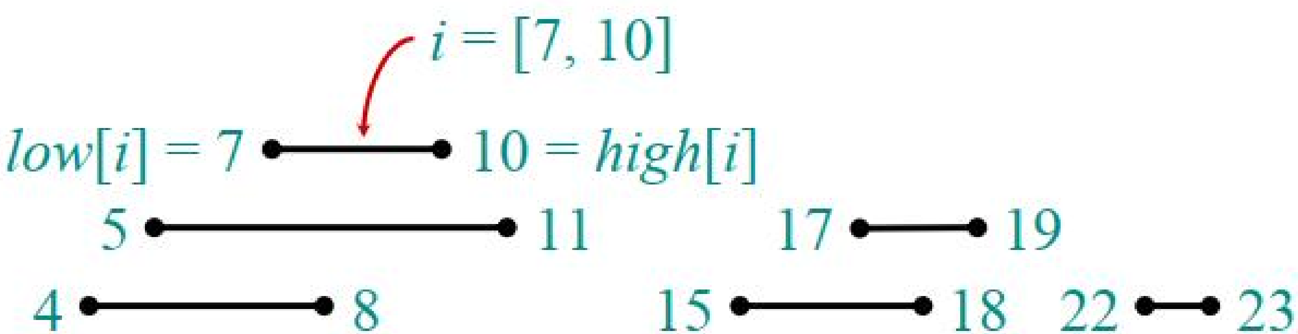
These steps are guidelines, not rigid rules.

Why Augmenting Data Structures



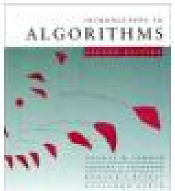
Interval trees

Goal: To maintain a dynamic set of intervals, such as time intervals.



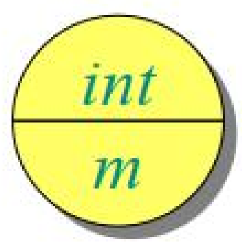
Query: For a given query interval i, find an interval in the set that overlaps i.

Why Augmenting Data Structures

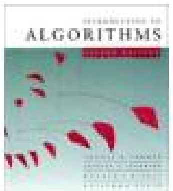


Following the methodology

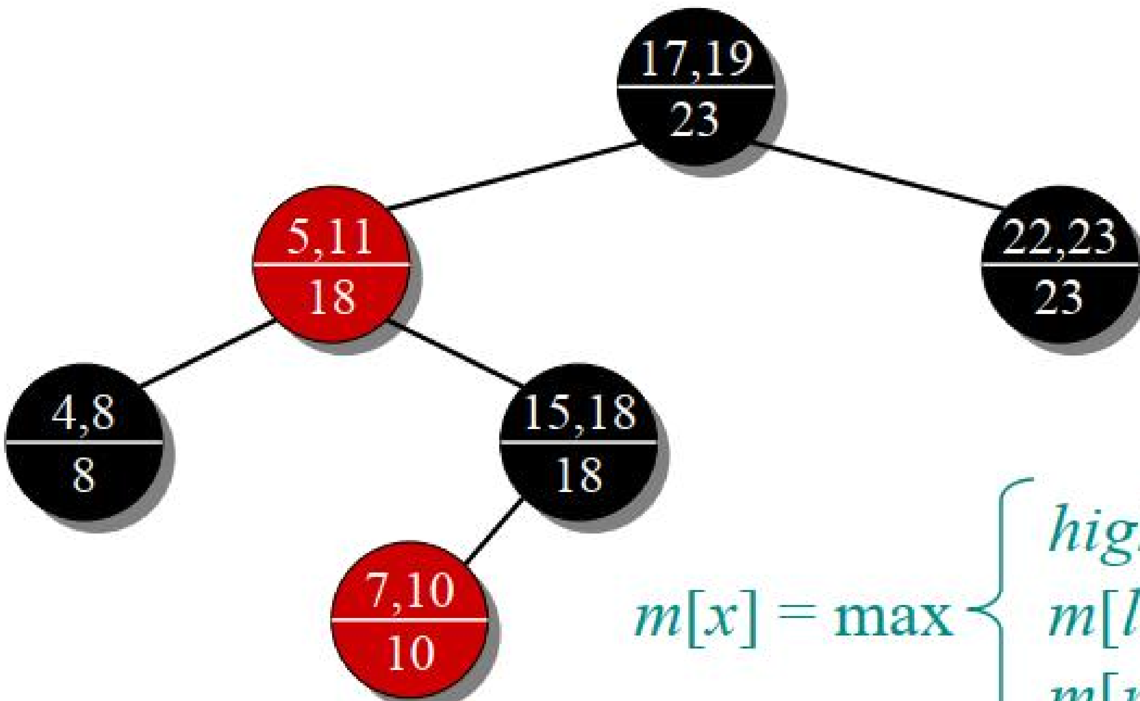
1. *Choose an underlying data structure.*
   - Red-black tree keyed on low (left) endpoint.
2. *Determine additional information to be stored in the data structure.*
   - Store in each node x the largest value $m[x]$ in the subtree rooted at x, as well as the interval $int[x]$ corresponding to the key.



Why Augmenting Data Structures

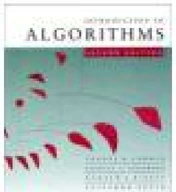


Example interval tree



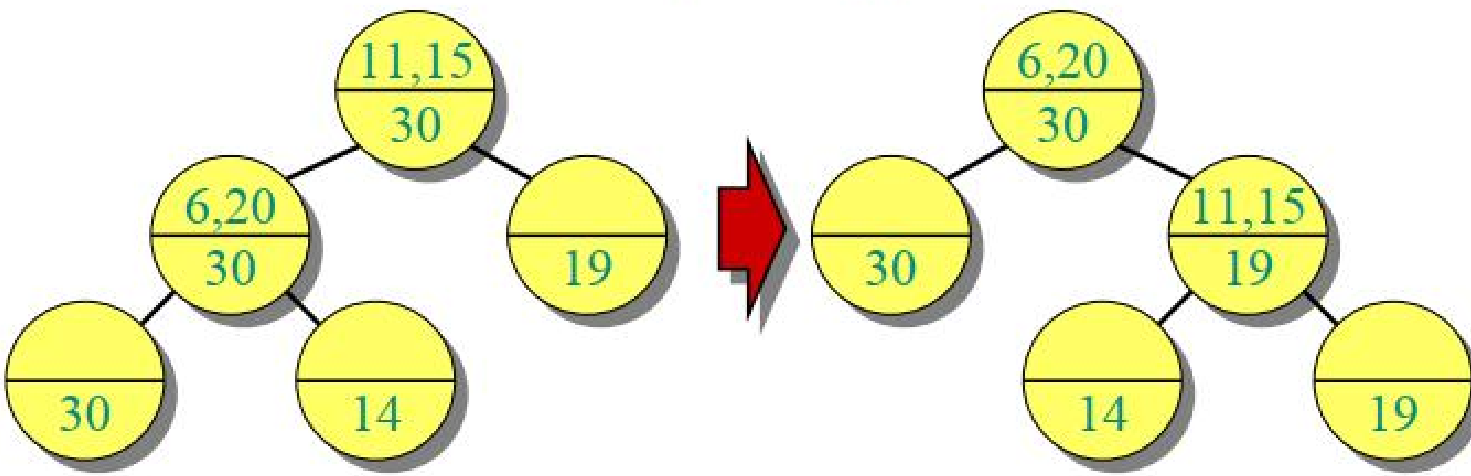
$$m[x] = \max \begin{cases} high[int[x]] \\ m[left[x]] \\ m[right[x]] \end{cases}$$

Why Augmenting Data Structures



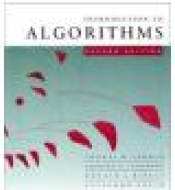
Modifying operations

3. *Verify that this information can be maintained for modifying operations.*
   - INSERT: Fix m's on the way down.
   - Rotations — Fixup $= O(1)$ time per rotation:



Total INSERT time $= O(\lg n)$; DELETE similar.

Why Augmenting Data Structures

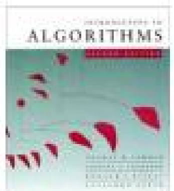


New operations

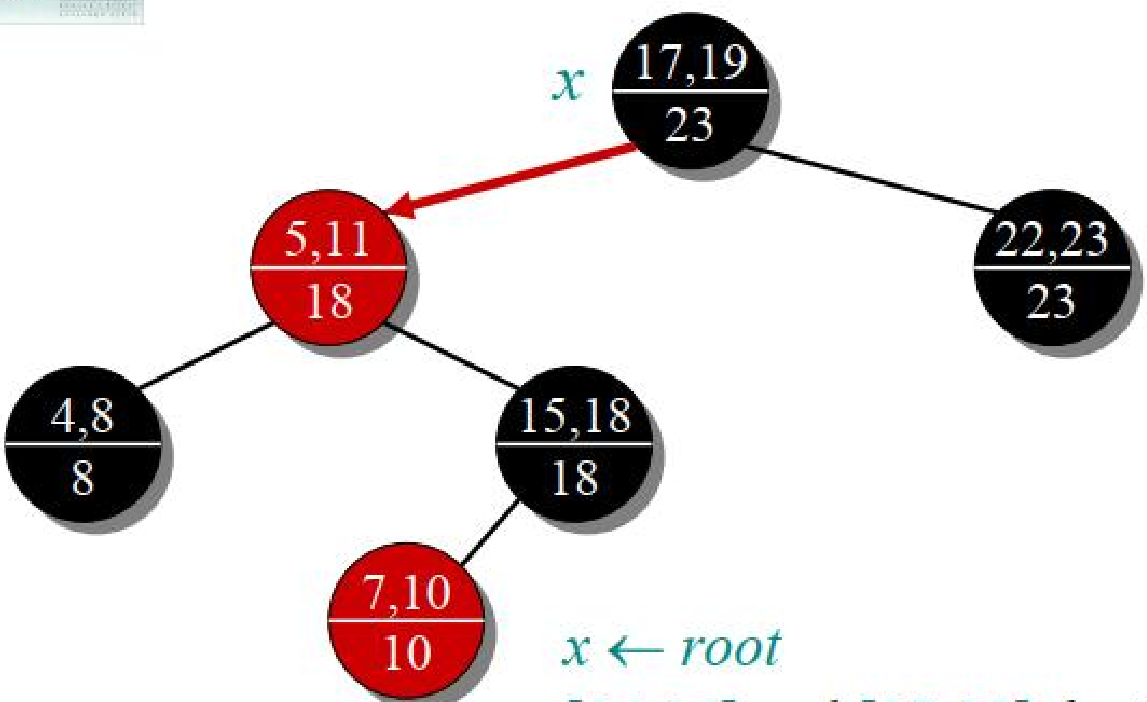
4. Develop new dynamic-set operations that use the information.

```
INTERVAL-SEARCH(i)
  x ← root
  while x ≠ NIL and (low[i] > high[int[x]]
                      or low[int[x]] > high[i])
     do ▷ i and int[x] don't overlap
        if left[x] ≠ NIL and low[i] ≤ m[left[x]]
           then x ← left[x]
           else x ← right[x]
  return x
```

Why Augmenting Data Structures



Example 1: INTERVAL-SEARCH([14,16])

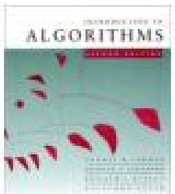


$x \leftarrow root$

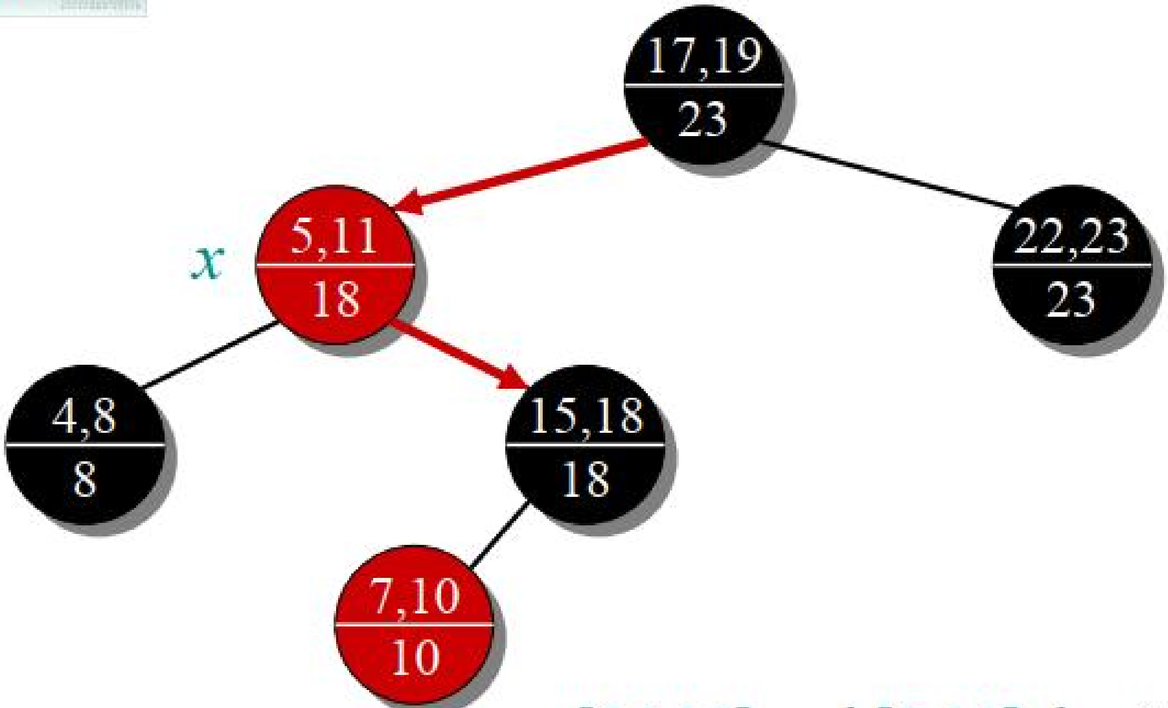
[14,16] and [17,19] don't overlap

$14 \leq 18 \Rightarrow x \leftarrow left[x]$

Why Augmenting Data Structures



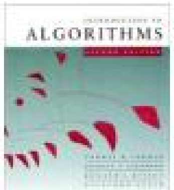
Example 1: INTERVAL-SEARCH([14,16])



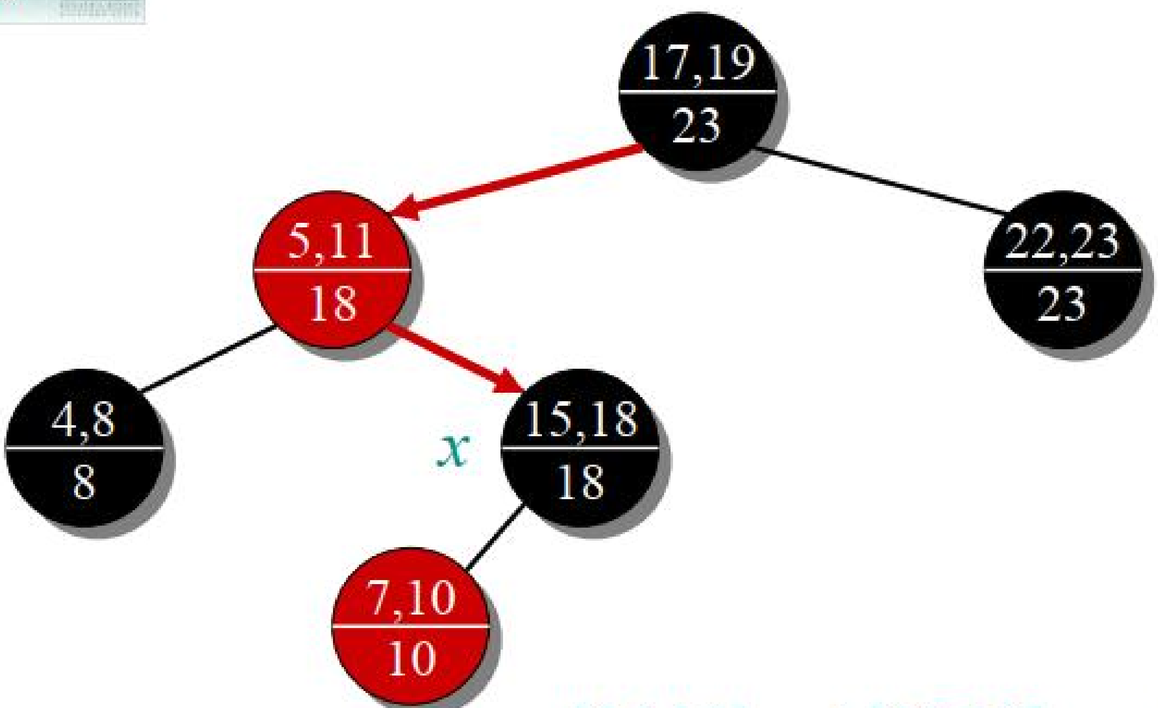
[14,16] and [5,11] don't overlap

$14 > 8 \Rightarrow x \leftarrow right[x]$

Why Augmenting Data Structures



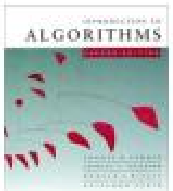
Example 1: INTERVAL-SEARCH([14,16])



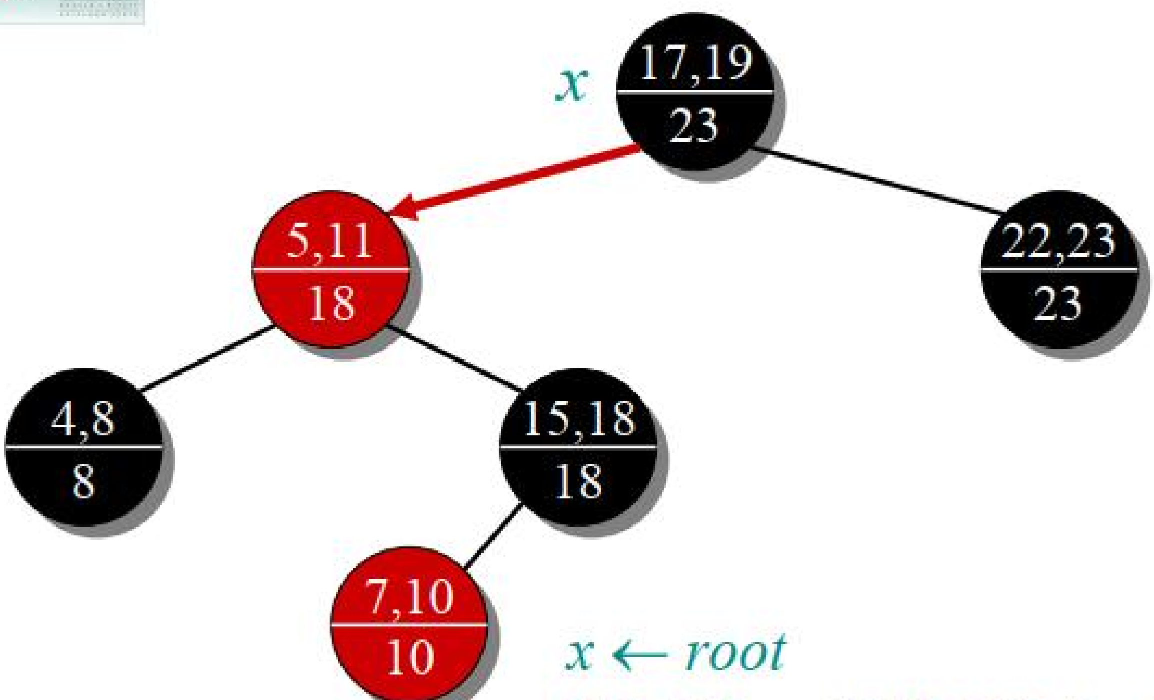
[14,16] and [15,18] overlap

return [15,18]

Why Augmenting Data Structures



Example 2: INTERVAL-SEARCH([12,14])

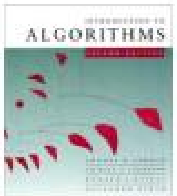


$x \leftarrow root$

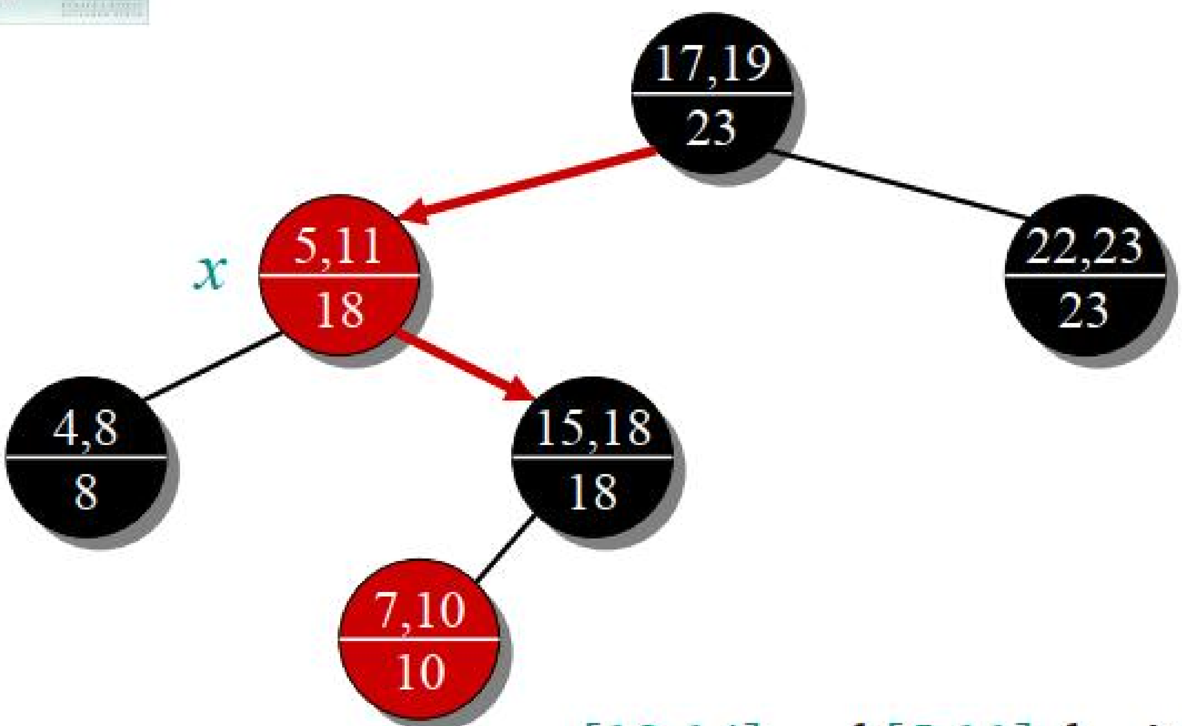
[12,14] and [17,19] don't overlap

$12 \leq 18 \Rightarrow x \leftarrow left[x]$

Why Augmenting Data Structures



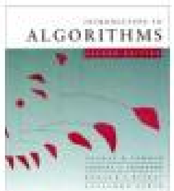
Example 2: INTERVAL-SEARCH([12,14])



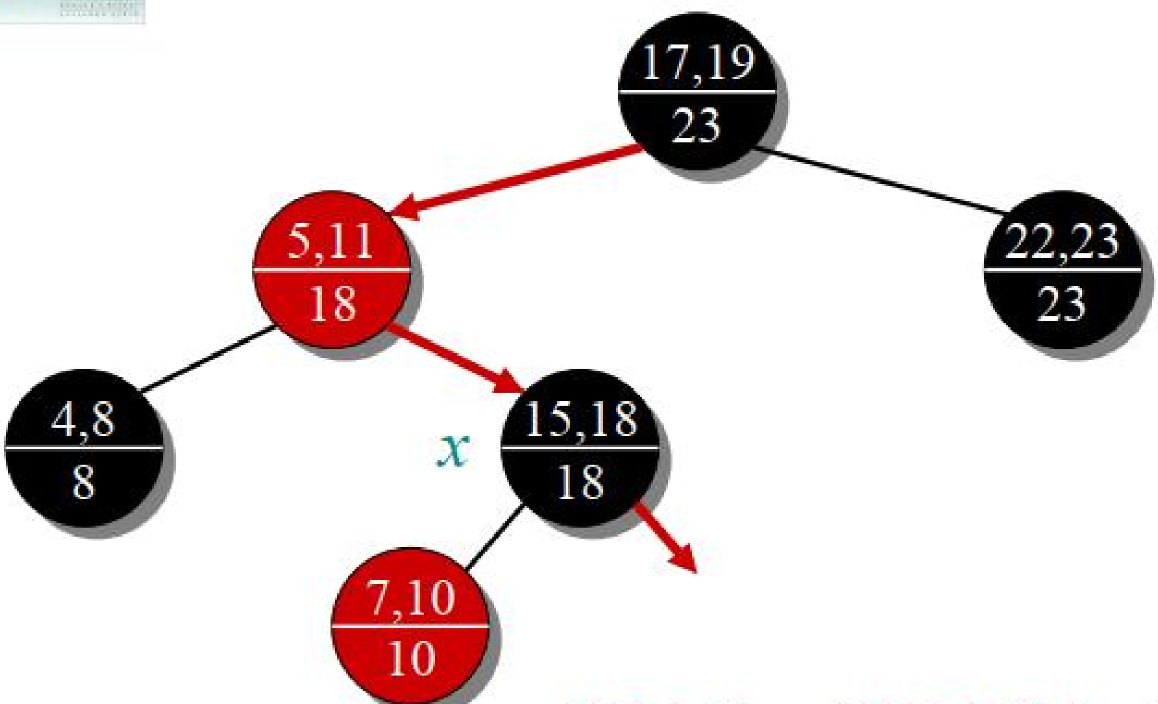
[12,14] and [5,11] don't overlap

$12 > 8 \Rightarrow x \leftarrow right[x]$

Why Augmenting Data Structures



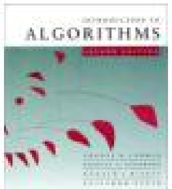
Example 2: INTERVAL-SEARCH([12,14])



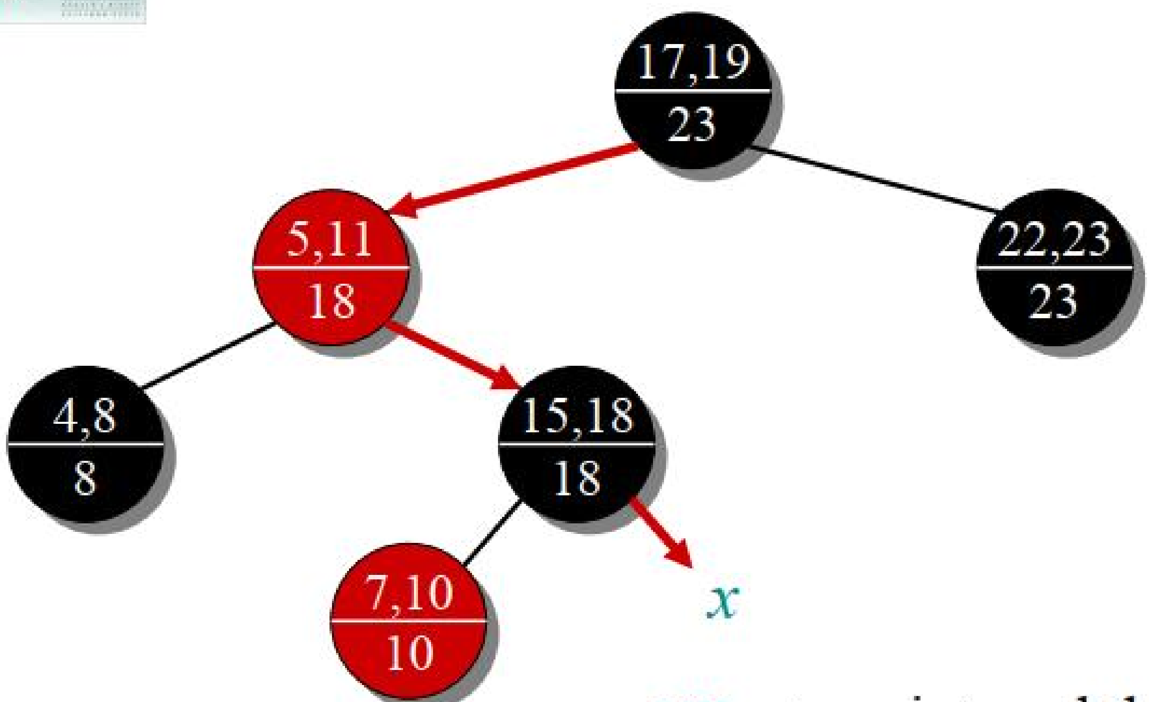
[12,14] and [15,18] don't overlap

$12 > 10 \Rightarrow x \leftarrow right[x]$

Why Augmenting Data Structures



Example 2: INTERVAL-SEARCH([12,14])

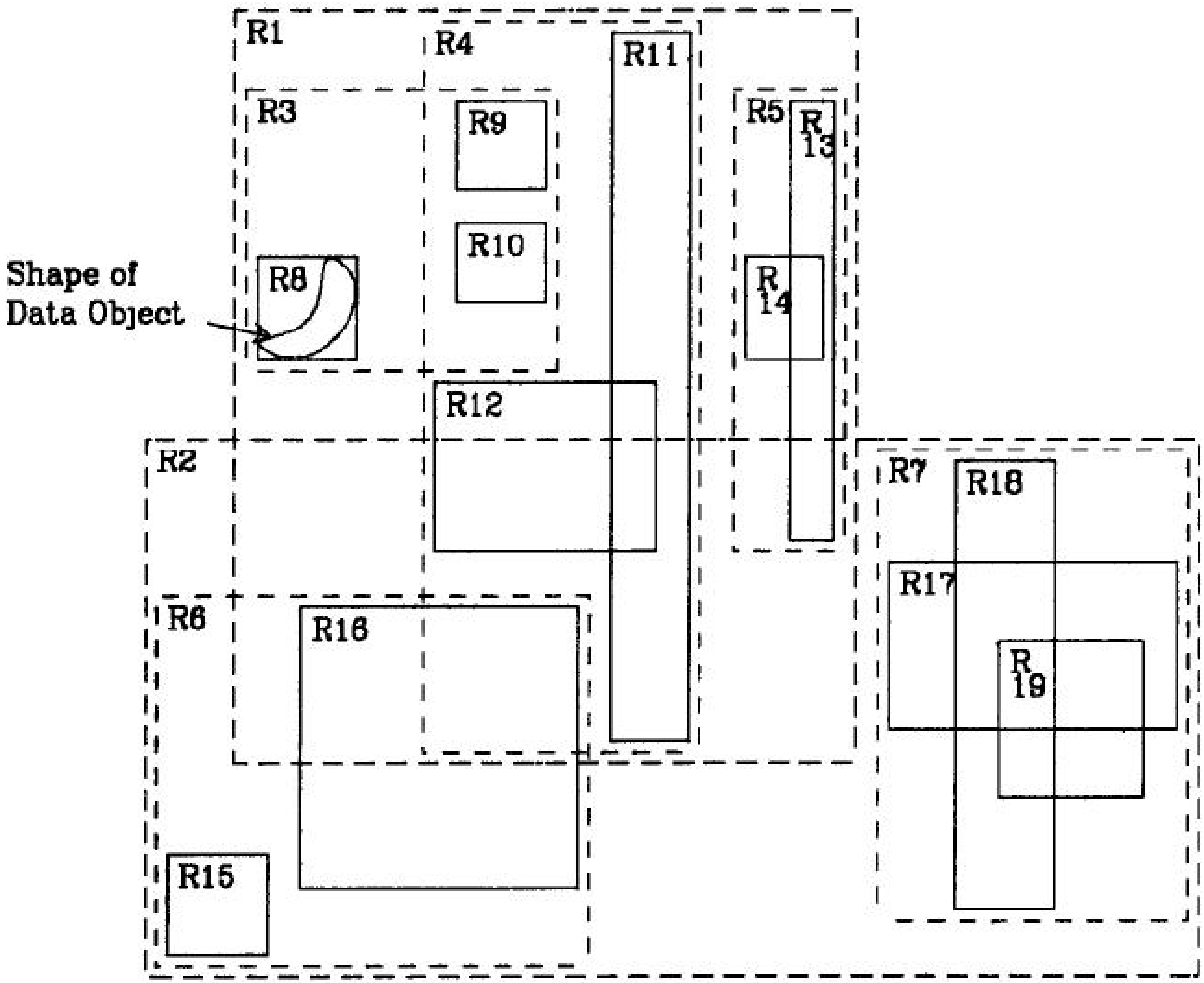


x = NIL $\Rightarrow$ no interval that overlaps [12,14] exists

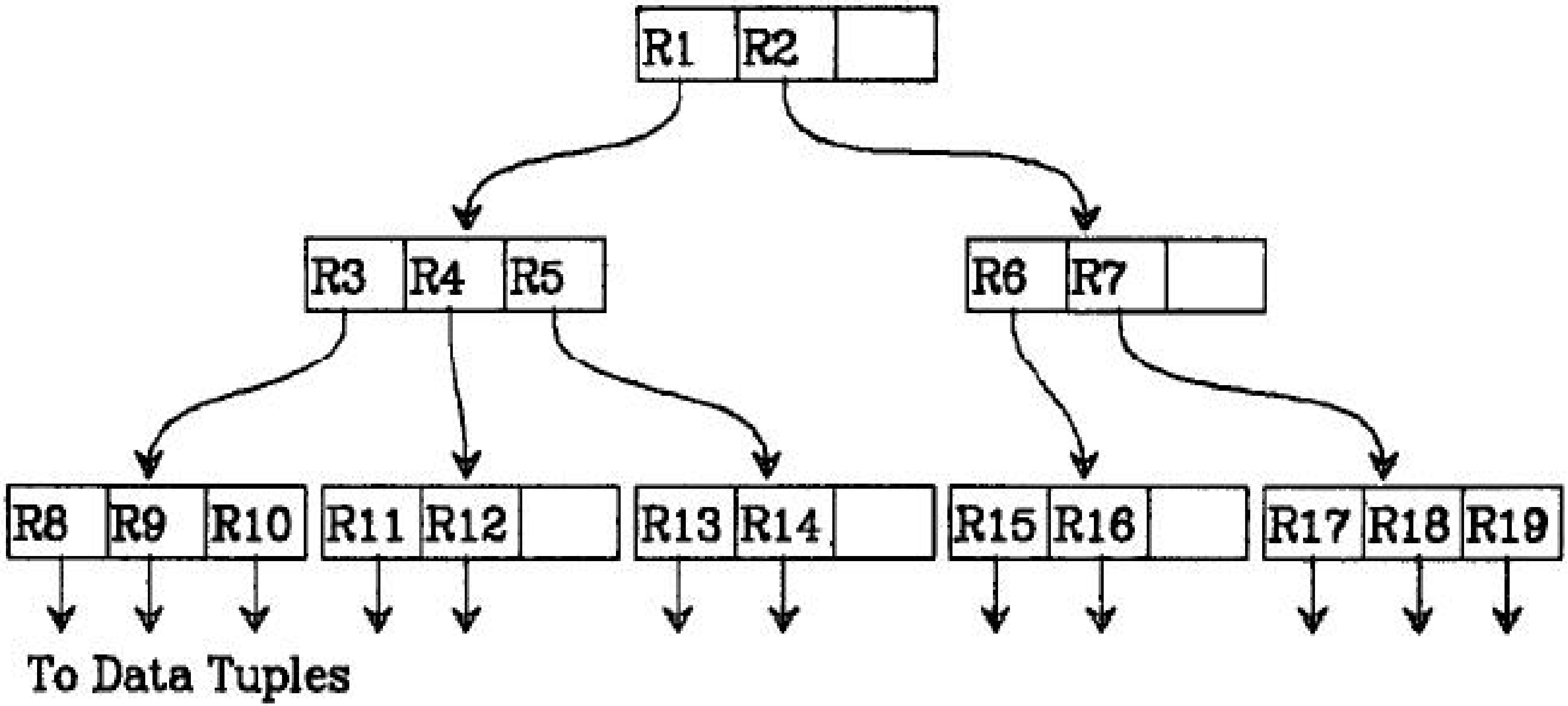
Outline

- **Why Augmenting Data Structures**
  - Access Pattern: Ranked Query
  - Non-Ordered Data: Data of Intervals
- **B+ based Augmenting Data Structures**
  - R-B+
  - Hilbert-Tree
- **Applications and Problems**

And R-tree

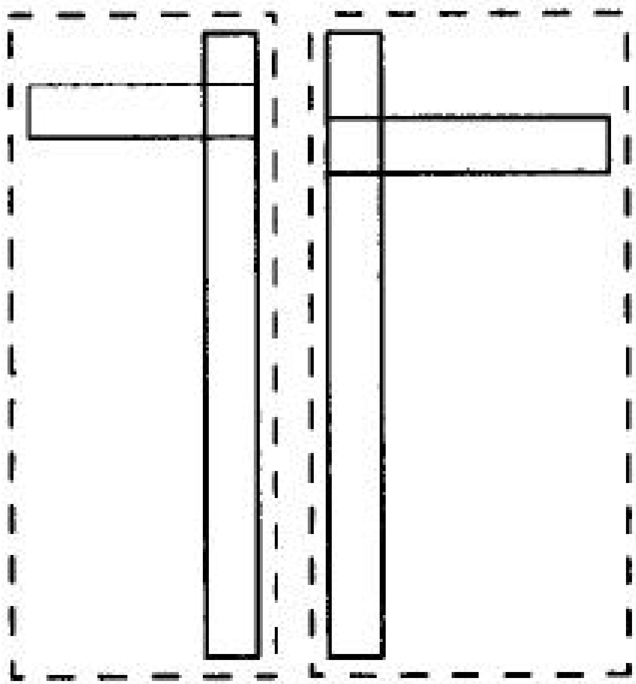


Shape of Data Object
R1
R4
R11
R3
R9
R5
R 13
R10
R8
R 14
R12
R2
R7
R18
R17
R6
R16
R 19
R15

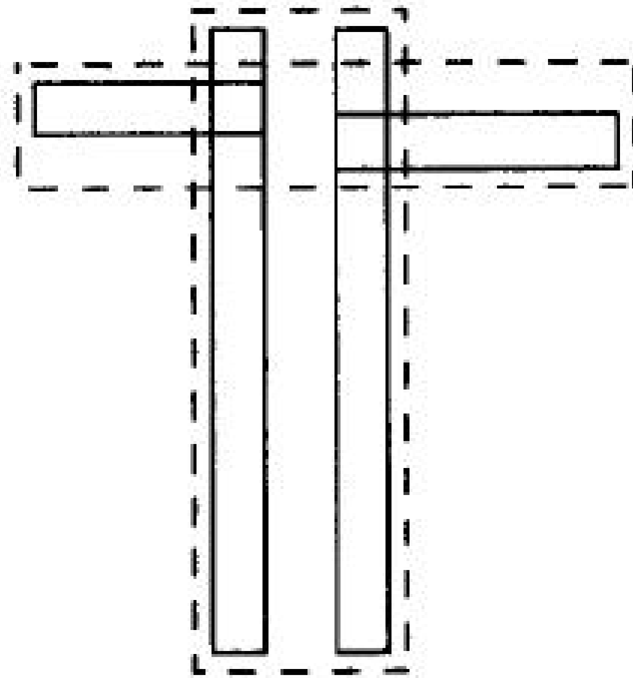


R-TREES: A DYNAMIC INDEX STRUCTURE FOR SPATIAL SEARCHING

Antonin Guttman
University of California
Berkeley



Bad split



Good split

And Hilbert R-tree

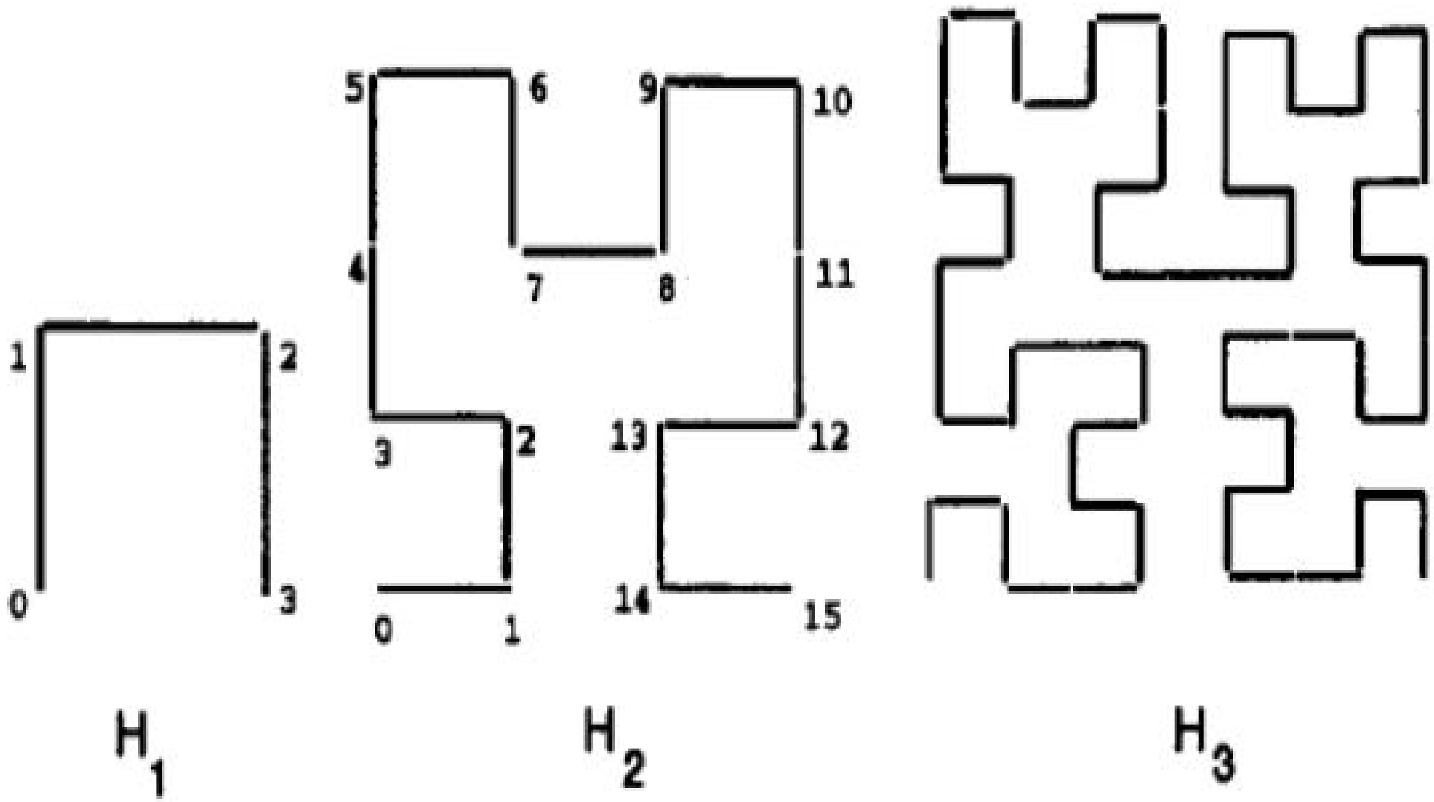
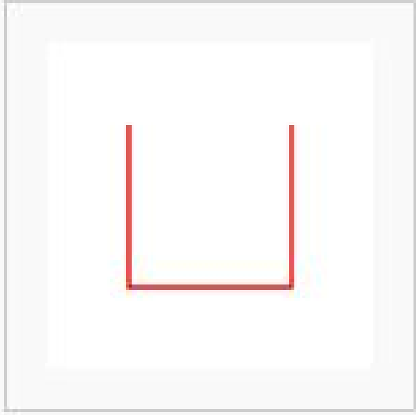
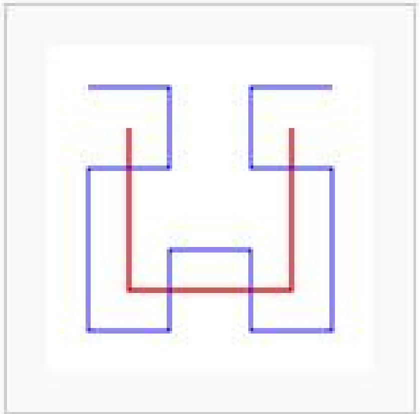


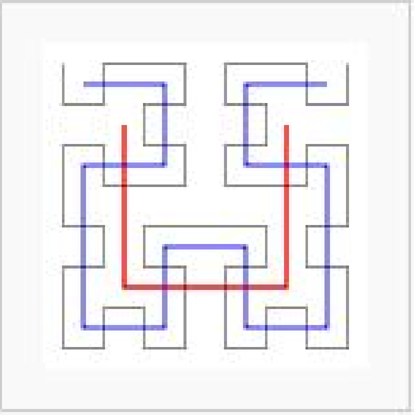
Figure 1: Hilbert Curves of order 1, 2 and 3



Hilbert curve, first order

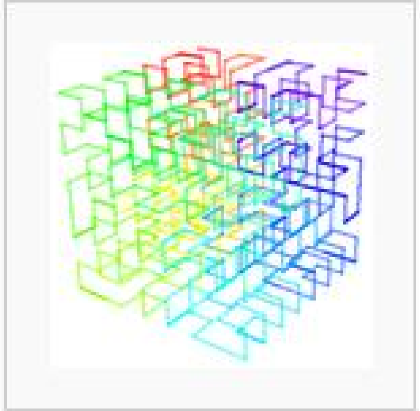


Hilbert curves, first and second orders

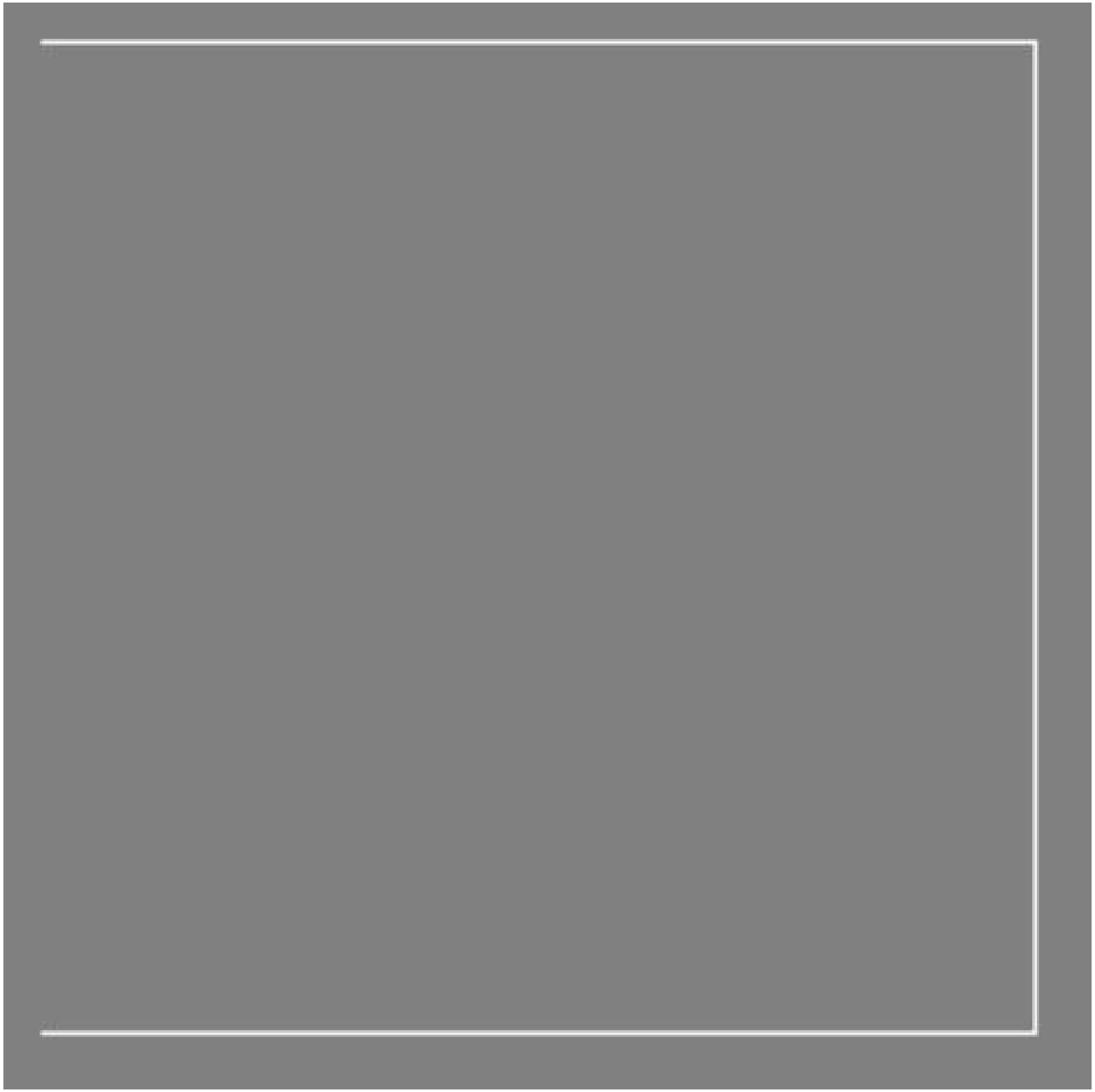


Hilbert curves, first to third orders

Hilbert curve in three dimensions



3-D Hilbert curve with color showing progression



Discover the power of custom layouts

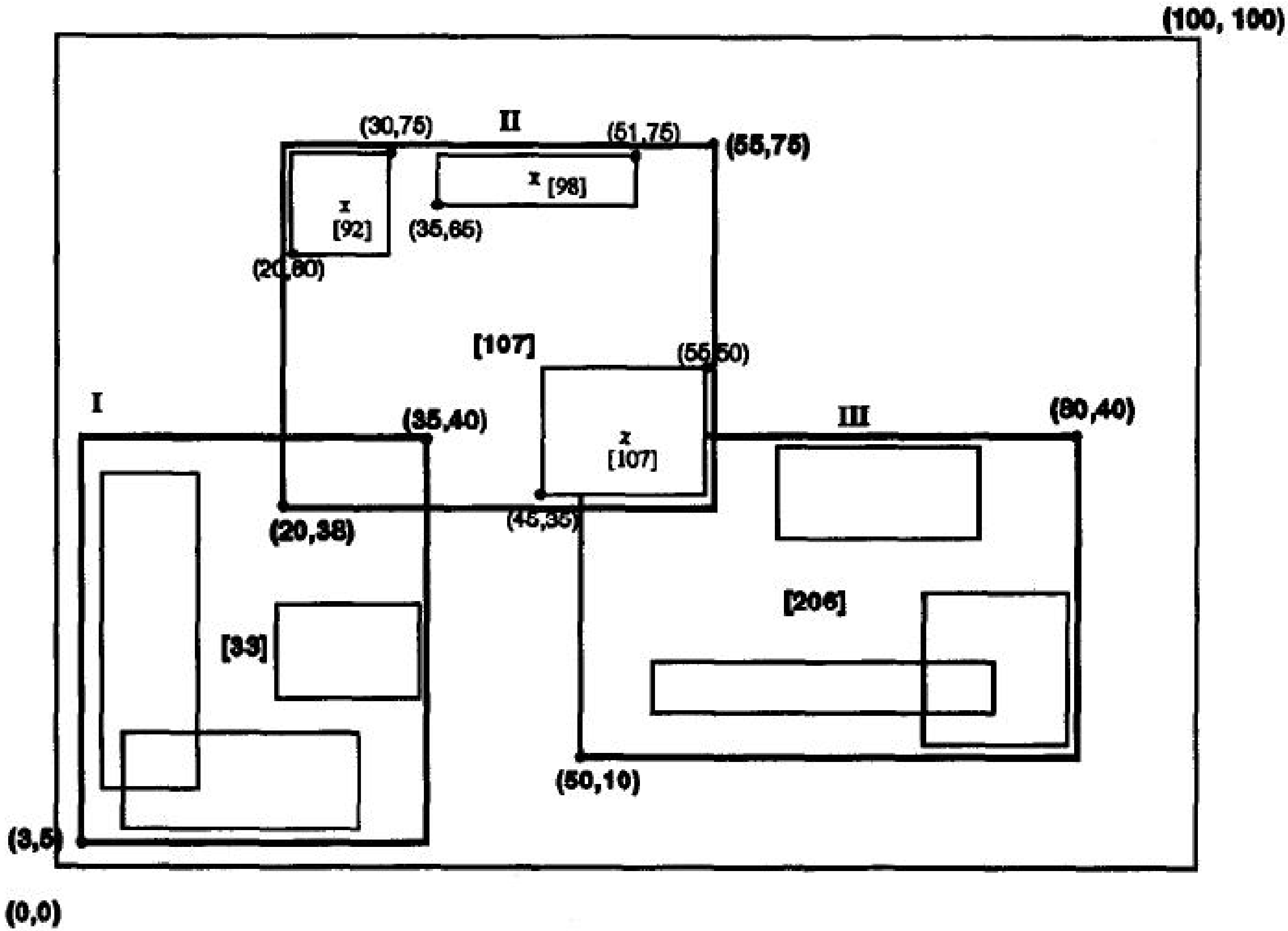


Figure 2: *Data rectangles organized in a Hilbert R-tree*

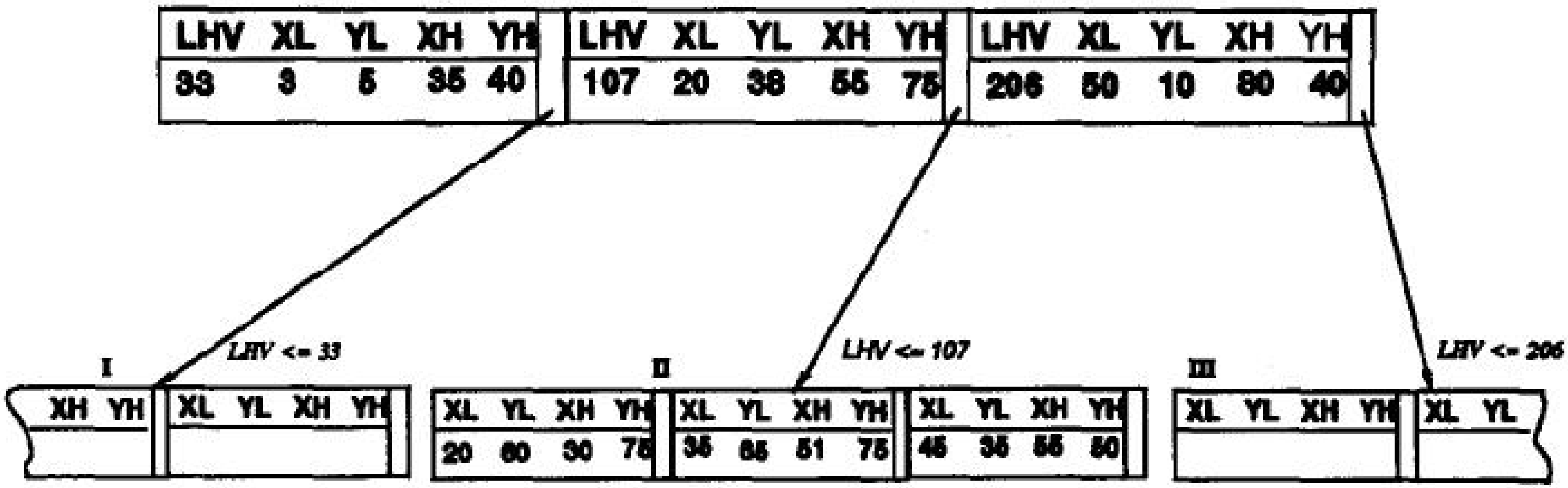


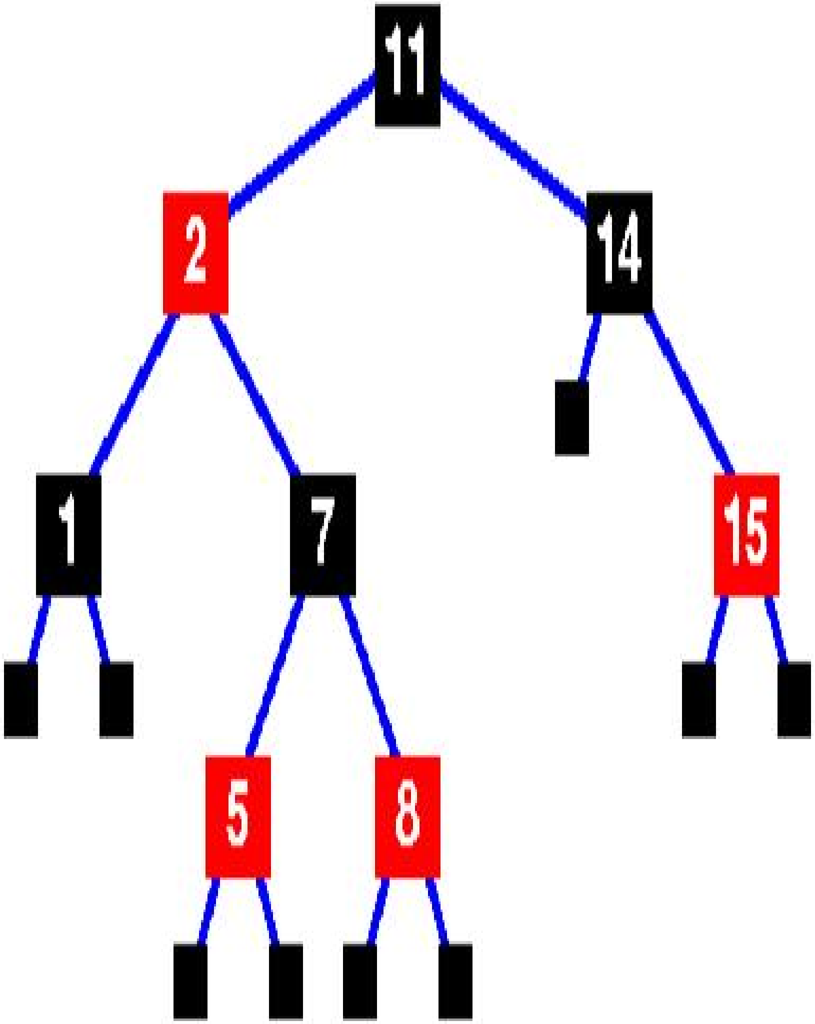
Figure 3: *The file structure for the previous Hilbert R-tree*

Outline

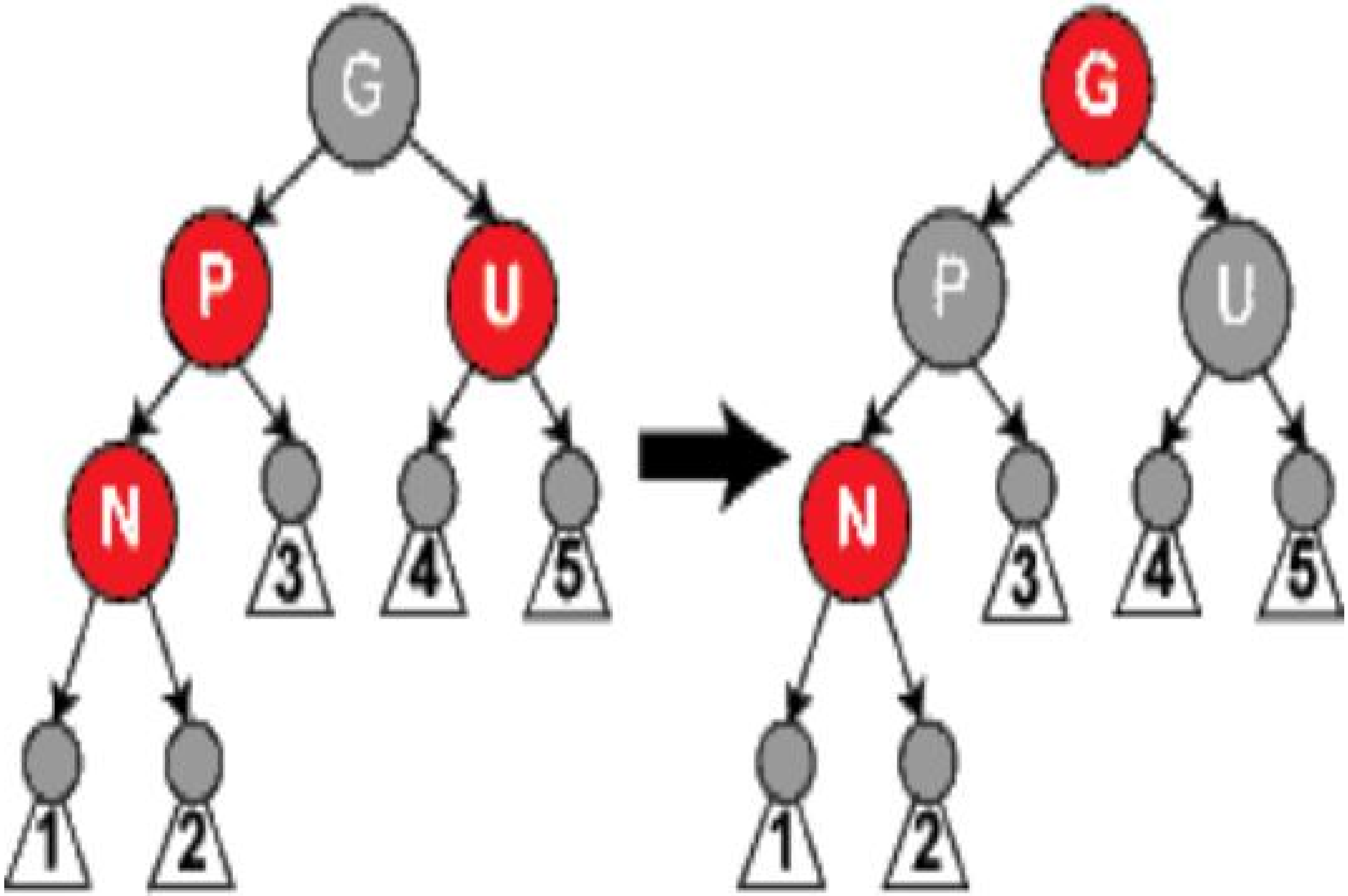
- **Why Augmenting Data Structures**
  - Access Pattern: Ranked Query
  - Non-Ordered Data: Data of Intervals
- **B+ based Augmenting Data Structures**
  - R-B+
  - Hilbert-Tree
- **Applications and Problems**

And Red Black-tree

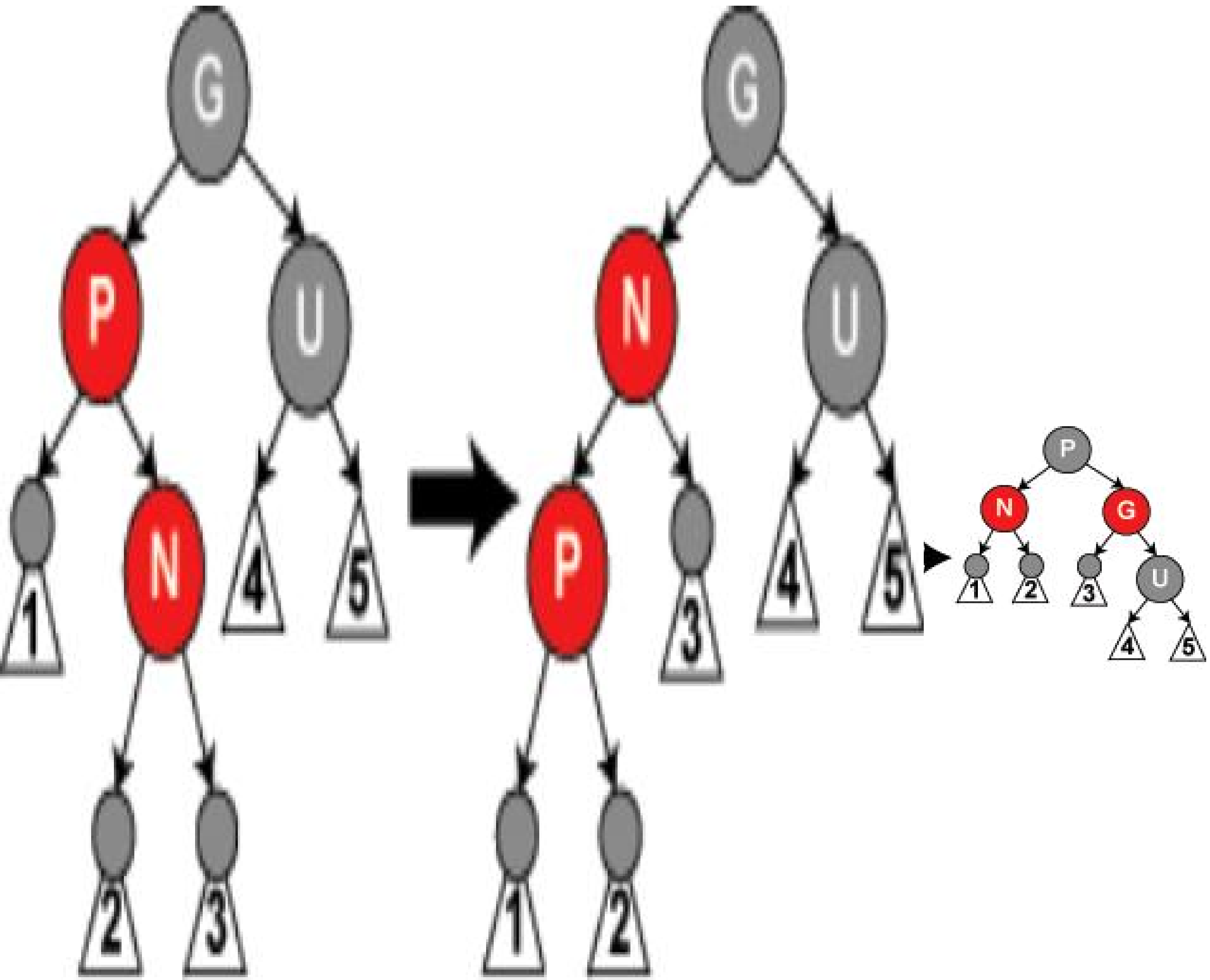
1. A node is either red or black.
2. The root is black
3. All leaves are black.
4. Both children of every red node are black.
5. Every simple path from a given node to any of its descendant leaves contains **the same number of black nodes.**



Insertion

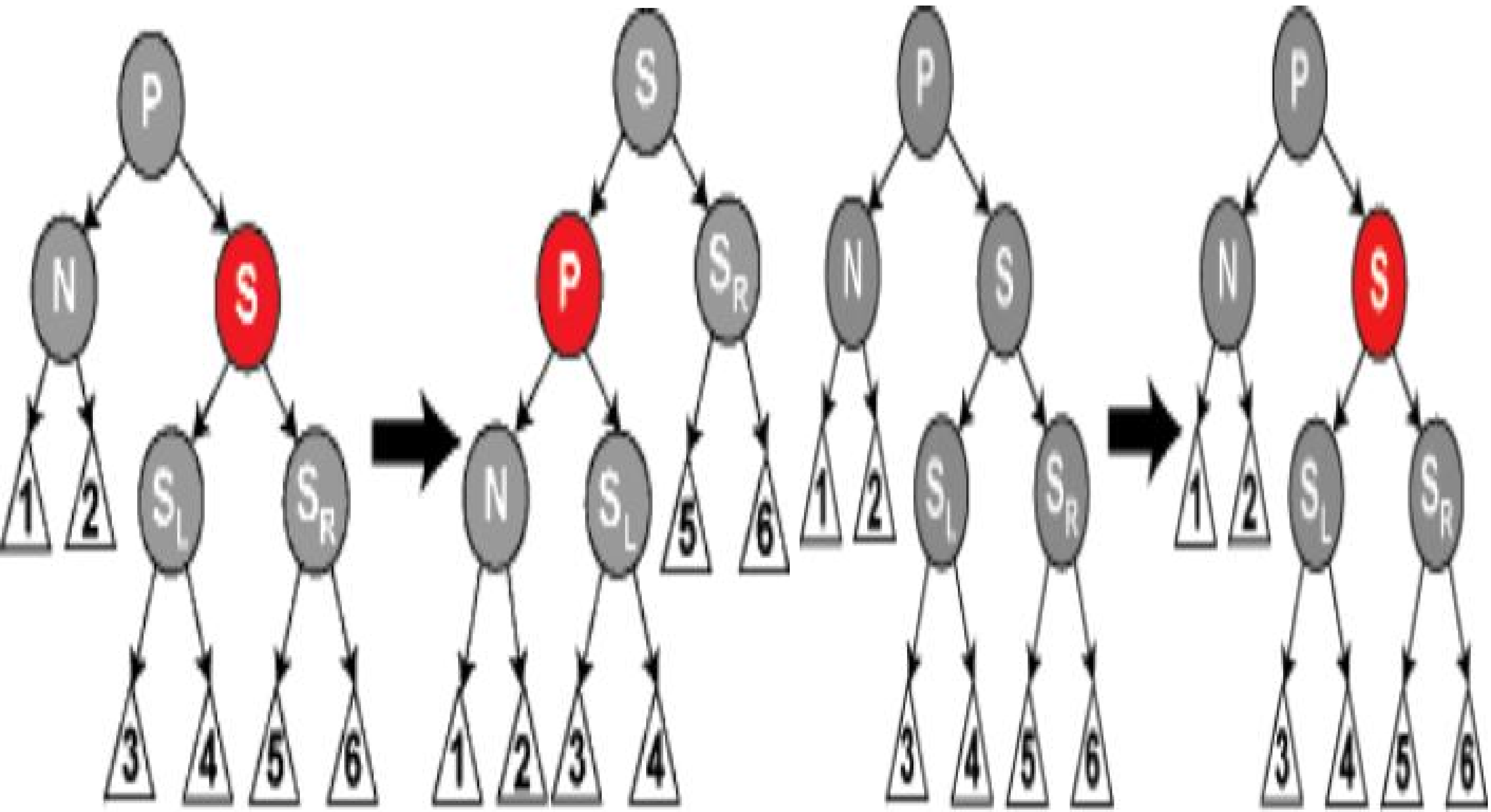


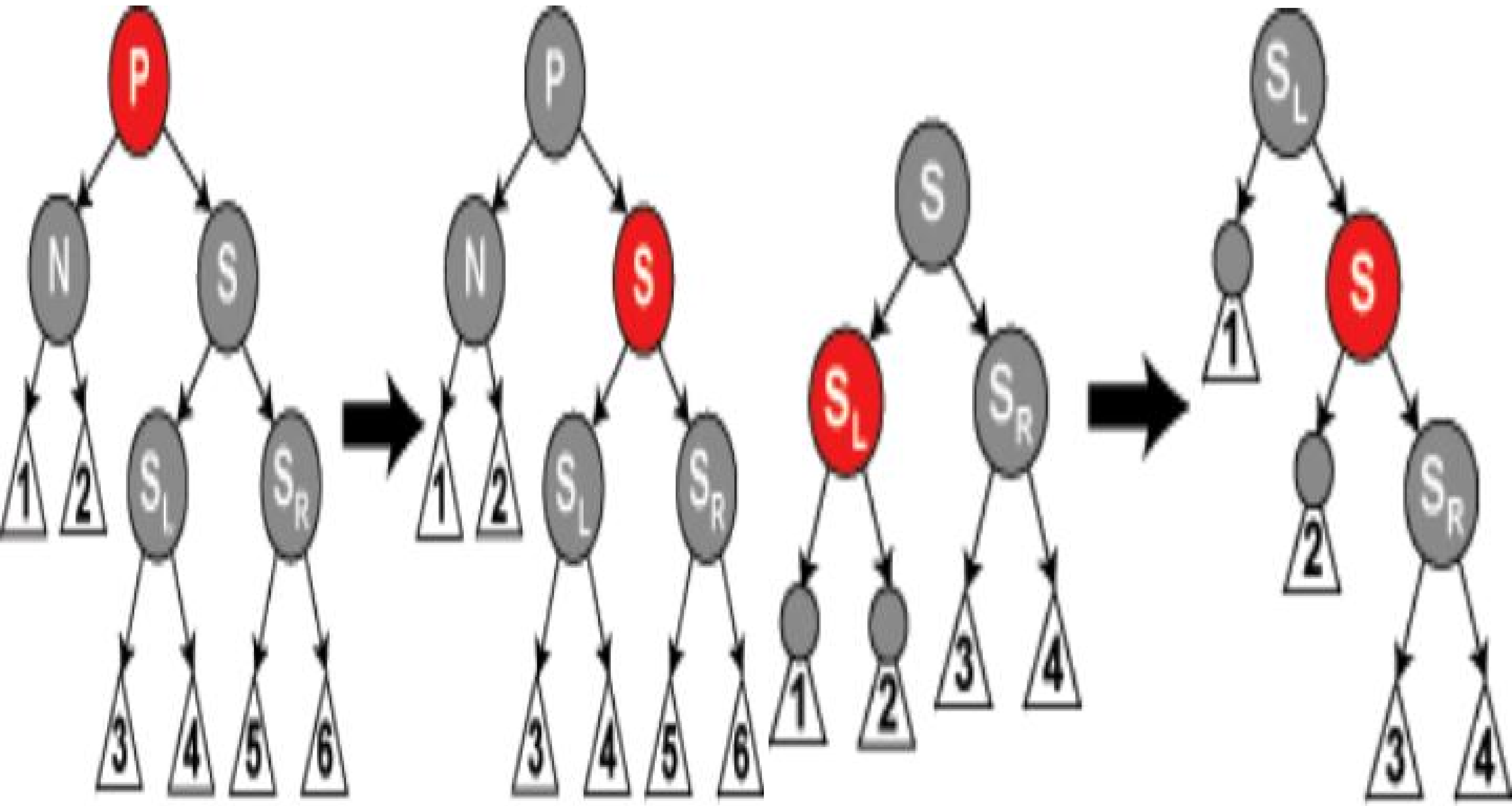
Insertion



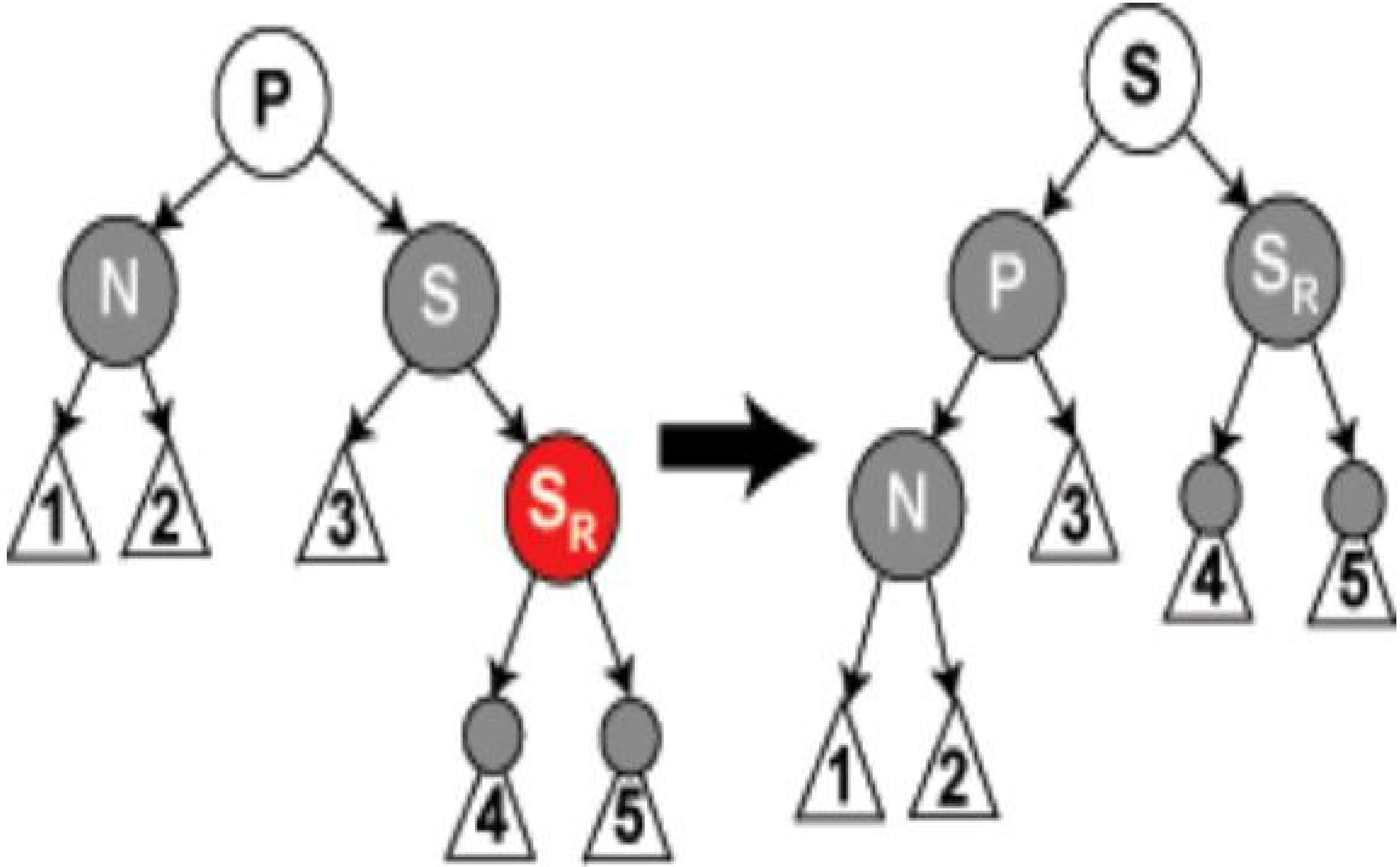
Deletion

1. this reduces to the problem of deleting a node with at most one non-leaf child.
   - deleting a red node
   - the deleted node is black and its child is red
   - both the node to be deleted and its child are black


P
N
S
S_L
S_R
1
2
3
4
5
6



Discover the power of custom layouts


P
N
S
1
2
3
S_R
4
5
S
P
S_R
N
3
4
5
1
2